Lab1: Simple Matrix Multiplication

徐蕊琦 1900013088

实验代码：

① 根据精度范围用 ap_ufixed 代替 double，并修改 testbench 和.h 文件：

```
#include "ap_fixed.h"
typedef ap_ufixed<8, 5> in_t;
typedef ap_ufixed<22, 16> data_t;

void mm(in_t A[I][K], in_t B[K][J], data_t C[I][J]);
```

② 用 memcpy 缓存 B 数组到本地，由于一次可以 load 两个值，比 64*64 的循环方式要快一倍：

```
in_t local_B[K][J];
```

```
  for (int i = 0; i < I; i++)
#pragma HLS pipeline II=1
      memcpy(&local_B[i], &B[i], 64 * sizeof(in_t));
```

③ 矩阵乘法使用的公式是 A[0][0]*B 的第一行 + A[0][1]*B 的第二行…… = C 的第一行

A[1][0]*B 的第一行 + A[1][1]*B 的第二行…… = C 的第二行

然后由于 DSP 限制同时只进行 16 个运算，即某个 A 的数乘 local_B 的 16 个数加到 local_C 的 16 个数里面，所以 local_B 和 local_C 都要 array_partition：

```
#pragma HLS array_partition variable=local_B dim=2 cyclic factor=16
#pragma HLS array_partition variable=local_C dim=2 cyclic factor=16

  for(int x = 0; x < I; x++){
      for(int j = 0; j < J; j++){
        for(int i = 0; i < I; i+=16){
#pragma HLS pipeline II=1
            for(int k = 0; k < 16; k++)
#pragma HLS unroll
              local_C[x][i+k] += A[x][j] * local_B[j][i+k];
        }
      }
#pragma HLS dataflow
      copy_C(local_C[x], C[x]);
  }
}
```

④ 由于 C 每计算完一行(一个 x 循环)就可以赋值给输出了，这与下一次循环无关，所以使用 dataflow 同时完成：

```
void copy_C(data_t* local_C, data_t C[J]){
#pragma HLS inline off
    for(int j = 0; j < I; j++){
#pragma HLS pipeline II=1
        C[j] = local_C[j];
    }
}
```

注意 BRAM 限制 20，所以手动更改 depth 为 1：

```
  data_t local_C[I][J];
#pragma HLS stream variable=local_C off depth=1
```

实验结果：

```
PS D:\pku\dasanxia\xinpian\assignment1> python test.py
Generate mat.txt to srcs : Finished
Create Vitis HLS Project, Add Files, Run C-Simulation and Synthesis : Finished
Check C-Simulation Result: Passed!
Check Synthesis Result:
| Latency =  18949 | BRAM_18K = 17 | DSP = 7 | FF = 2 | LUT = 6 |
Score is 100. Congratulations!
PS D:\pku\dasanxia\xinpian\assignment1>
```